

# JS Level 0



# Введение



# Введение

Сегодняшняя наша лекция будет посвящена JS – мы начнём знакомиться с этим языком и будем выяснять, что с помощью него можно делать.



**JS**



# JavaScript

**JavaScript (JS)** – язык программирования, позволяющий взаимодействовать с загруженной в браузер веб-страницей. Это не единственное применение JS, но для нас будет ключевым (разработку серверных приложений мы будем рассматривать на Level 3).



# JavaScript

JavaScript (также, как и CSS) нужно встроить на страницу, чтобы браузер его обработал и запустил.

Здесь важно отметить, что работает всё примерно так же – браузер встречает нужный элемент (в CSS был элемент `style` и атрибут `style`) и применяет те инструкции, которые там написаны.



# JavaScript

Варианты встраивания:

1. Элемент `script`, ссылающийся на внешний файл (аналог `link rel="stylesheet"`):

```
<body>
| <script src="app.js"></script>
</body>
```

2. Элемент `script` с кодом (аналог элемента `style`):

```
<body>
| <script>
| | alert('js worked!');
| </script>
</body>
```

3. Атрибуты элементов (аналог атрибута `style`, но специфический):

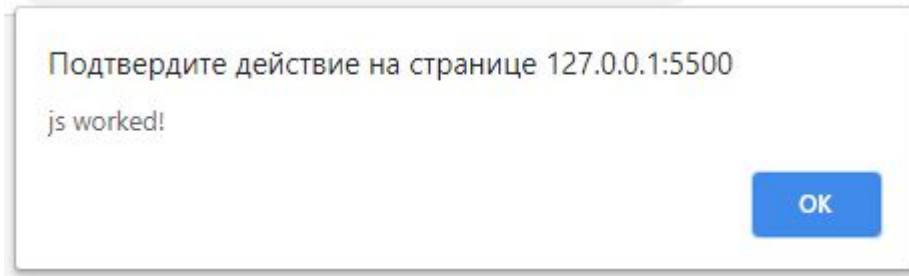
```
<button onclick="alert('js worked!');">Клики на мне</button>
```



# JavaScript

Мы для простоты будем использовать второй вариант.

После обновления страницы вы увидите всплывающее сообщение (почему так делать не надо, мы с вами ещё поговорим):





# JS

Q: что умеет JS?

A: здесь нужно разделить вопрос на два:














1. Что умеет JS в браузере
2. Что он умеет вне браузера

Пока разберёмся с браузером. В браузере JS может очень много: создавать/изменять/удалять элементы, менять их стили, загружать данные (например, когда можно загружать фотографии на сайт с помощью JS). И это только малая часть: можно записывать аудио/видео, организовывать видеозвонки из одного браузера в другой, демонстрировать экран другому человеку и т. д.



# Список возможностей

## Permissions

	<a href="#">Reset permissions</a>
 Location	Allow ▼
 Camera	Allow ▼
 Microphone	Allow ▼
 Motion sensors	Allow (default) ▼
 Notifications	Allow ▼
 File editing	Ask (default) ▼
 HID devices	Ask (default) ▼
 Clipboard	Ask (default) ▼
 Payment Handlers	Allow (default) ▼
 Insecure content	Block (default) ▼
 Augmented reality	Ask (default) ▼
 Virtual reality	Ask (default) ▼
 Your presence	Ask (default) ▼

Некоторые возможности достаточно опасны (например, возможность записывать видео с вашей камеры), поэтому браузер будет явно у вас спрашивать, разрешить ли странице доступ к камере.



# JS

Работать с этими возможностями (запись аудио/видео/чаты/звонки) мы будем на более старших курсах ("секретных уровнях JS"), пока же нам важно просто познакомиться с этим языком.

В первую очередь нас интересует возможность писать приложения (или программы), которые будут выполняться в браузере. И мы сразу с вами будем рассматривать реальные задачи, чтобы научиться их решать\*.

Примечание\*: на более старших курсах мы будем писать сервера на Node.js.



# JS

**Ключевое:** если вы захотите написать какую-нибудь более-менее сложную анимацию (фото/видео галереи), или реагировать на поведение пользователя (показывать всплывающие окна, подсказки, уведомлять пользователя об ошибках) – вам не обойтись без JS.

Именно поэтому мы и будем его изучать. Самое главное – начать с простого: обычных чисел и операций с ними. Потому что делая ту же самую анимацию, мы будем вычислять позицию элемента в пикселях (а это числа).



# JS

Одна из задач – виджет перевода денег. Виджет – это такой небольшой самостоятельный компонент на веб-странице, который умеет делать хорошо одну задачу. Так вот наш виджет выглядит следующим образом:

Валюта	Сумма	от 5 до 4000 TJS
TJS	<input type="text" value="10000"/>	<input type="button" value="X"/>
Курс конвертации	1 ₸ = 0,147 сомони	
Сумма к зачислению	68 027,21 ₸	
Комиссия alif mobi	1.0%	
Итого к оплате	10 100,00 сомони	
<input type="button" value="ДАЛЕЕ"/>		



# JS

Ключевая идея: пользователь вводит сумму в поле сумма, автоматически высчитываются все значения, и, если они пользователю нравятся, то он нажимает на кнопку "Далее" и переводит средства.

Валюта	Сумма	от 5 до 4000 TJS
TJS	10000	X
Курс конвертации	1 ₸ = 0,147 сомони	
Сумма к зачислению	68 027,21 ₸	
Комиссия alif mobi	1.0%	
Итого к оплате	10 100,00 сомони	

ДАЛЕЕ



# JS

Первое, чему мы должны научиться – это декомпозиция задач (разбиваем задачи на мелкие подзадачи). Несмотря на то, что виджет всего один, сразу можно выделить целых три подзадачи:

1. Визуальная составляющая: как пользователь будет вводить данные, как вы будете выводить ему результат
2. Алгоритм: формулы, расчёты (деньги нужно ещё конвертировать)
3. Взаимодействие с сервером (деньги пользователя не хранятся в браузере, они хранятся на счету в банке)

→ можем сделать уже сейчас



# Формулы

В принципе, всё достаточно просто: нам (как в школе) просто нужны формулы, по которым мы будем считать.

В реальных проектах вам могут давать готовые формулы и примеры расчёта по этим формулам (это идеальный вариант). Но бывает и так, что вам придётся самим эту формулы "вывести".

В нашем случае формулы расчёта будут такие:

- для суммы к зачислению:  $10\,000 / 0.147 = 68\,027.21$
- для суммы платежа:  $10\,000 * 1.01 = 10\,100$





# Что такое язык программирования?

До этого мы с вами обсуждали HTML – и говорили, что это язык, предназначенный для описания документов.

В нём есть специальные элементы, со своими атрибутами, правилами вложенности и т.д.

Но что же такое язык программирования? Язык программирования – это специальный язык, который представляет нам возможность заставить производить компьютер какие-то действия, например, считать (что нам и нужно).



# Формула

Если мы говорим про "считать", то первая аналогия – это калькулятор.

Давайте попробуем записать всё как в калькуляторе и попробовать запустить:

```
<body>
  <script>
    | 10000 / 0.147
  </script>
</body>
```

Обратите внимание, в числе пробелы не пишутся, и в качестве разделителя дробной и целой части выступает . (точка)

Запускаем Live Server, открываем нашу страницу и... ничего не происходит (или мы просто не видим, что что-то происходит).

Давайте разбираться.



# Формулы

В браузер встроен "движок" (engine) JS, который занимается тем, что разбирает те инструкции, которые мы написали и выполняет их.

Для простоты пока будем считать, что как только браузер обнаруживает элемент `script`, он подключает этот движок для разбора и выполнения инструкций в нём.

На данный момент нам важно увидеть, что это действительно происходит.



# Debugger



# Debugger

Для того, чтобы увидеть, как движок JS исполняет наш файл (и исполняет ли вообще), в Developer Tools встроен специальный инструмент, который называется **Debugger** (отладчик).

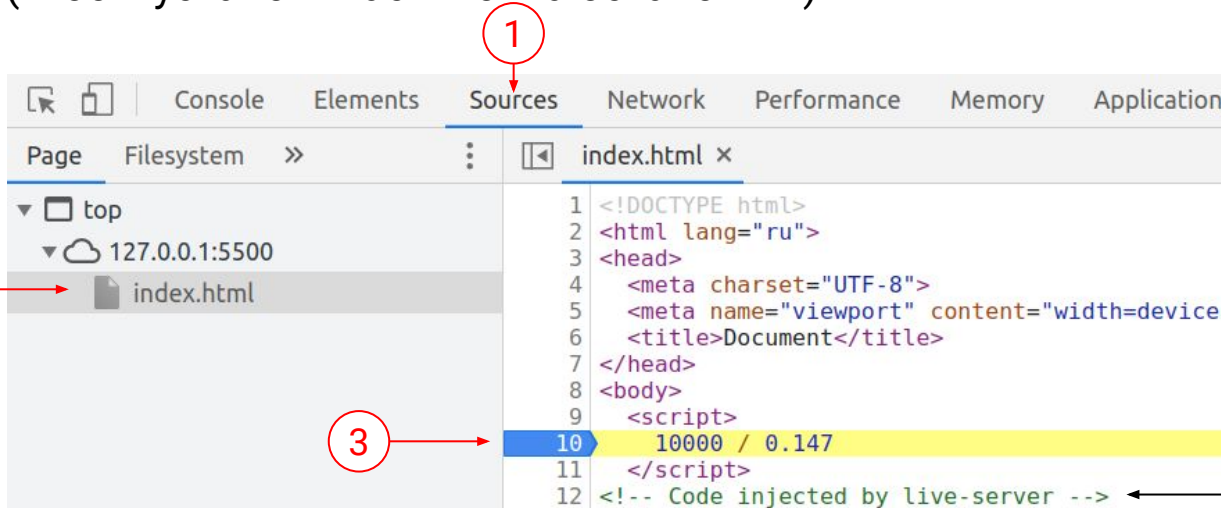
Отладчик – это специальный инструмент, который позволяет перевести движок JS в режим пошагового выполнения. При этом мы можем смотреть, что и как выполняется.

Далее для краткости мы не будем говорить "движок JS", а будем просто говорить браузер.



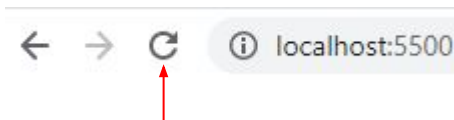
# Debugger

Нажимаем **F12** и открываем вкладку **Sources** (1), в боковой панели выбираем файл **index.html** (2) и на поле с номерами строк кликаем левой кнопкой мыши один раз (чтобы установилась "точка остановки"):



ниже будет код, который вставляет LiveServer, чтобы обновлять нашу страницу. Не обращайте на него внимания.

Точка остановки – это строка, на которой остановится выполнение. Для того, чтобы её активировать, нужно перезагрузить страницу (**F5**):

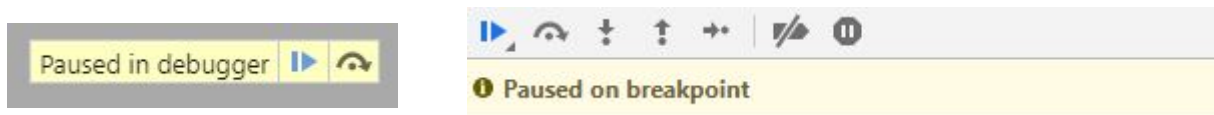


# Debugger

После обновления строка с формулой подсветится (это значит, что браузер ещё эту строку не выполнил):

```
8 <body>
9 <script>
10 10000 / 0.147
11 </script>
12 <!-- Code injected by live-server -->
```

Кроме того, на самой странице (и в панели справа появятся кнопки управления):



Нас будут интересовать только две:

- ▶ продолжить дальше, пока не встретится следующая точка остановки
- ↻ сделать один "шаг" вперёд



# Debugger

Нажмите на шаг вперёд и увидите, что браузер установит курсор в конец нашей "формулы" (на `</script>`):

```
9 | <script>  
10 | 10000 / 0.147  
11 | </script>
```


Это значит он вычислил результат нашего выражения. Вы можете попросить это сделать его для вас ещё раз и посмотреть на результат, выделив всё выражение (выделяйте так же, как вы выделяете обычный текст):

```
7 | </script>  
8 | <body>68027.21088435374  
9 | <script>  
10 | 10000 / 0.147  
11 | </script>
```





# Debugger

Если вы нажмёте ещё раз на шаг вперёд, то попадёте в код, вставленный LiveServer. Поэтому просто уберите точку остановки (кликните ещё раз на боковое поле) и нажмите на кнопку  .

Мы с вами посмотрели, что браузер действительно что-то делает с нашим выражением, но куда девается результат?

На самом деле, всё достаточно просто – поскольку мы ничего не делаем с результатом, браузер просто про него "забывает" (это, конечно, упрощённое описание, но нам подходит).



# alert

Мы уже использовали конструкцию `alert` для вывода текста во всплывающем окне. Давайте попробуем повторить этот же трюк:

```
<script>  
| alert(10000 / 0.147);  
</script>
```



127.0.0.1:5500 says

68027.21088435374

OK



# Как это работает?

Так же, как в математике, в JS есть понятие функций и приоритета. Например, если вы вспомните математику, то там были выражения вроде  $\sin(x)$ . Если мы перенесём на наш пример, то `alert(10000/0.147)` аналогично, например,  $\sin(3.14 / 2)$ .

Так же, как в математике, сначала вычисляет то, что в скобках, а потом вызывается функция с результатом:

1. Шаг 1:  **$10000/0.147 = 68027.21088435374$**
2. Шаг 2: **`alert(68027.21088435374)`**



# alert

Несмотря на то, что этот способ работает – он считается крайне дурным тоном, кроме того обладает побочными эффектами, например "замораживает" исполнение любого другого кода в открытой вкладке браузера.

Давайте посмотрим на альтернативы.



**console.log**



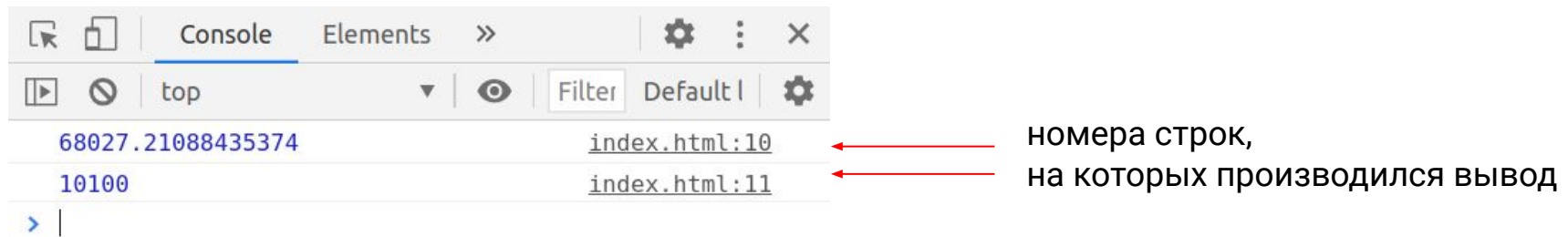
# console.log

В качестве альтернативы достаточно часто используют `console.log`. Почему в названии присутствует точка мы с вами поговорим чуть позже, пока же для нас важно, что этот вызов позволяет выводить значения в консоль браузера:

```
<script>  
  console.log(10000 / 0.147);  
  console.log(10000 * 1.01);  
</script>
```

← в конце строки ставится ;

Сама консоль расположена на вкладке **Console** в Developer Tools:



Консоль браузера (не **console.log**) – достаточно мощный инструмент, им обязательно нужно научиться пользоваться.



# Промежуточные итоги

Мы с вами посмотрели как пользоваться отладчиком. На последнем примере увидели, что браузер исполняет наш код по строкам сверху-вниз и немного поговорили о том, как всё работает.

Но ключевая проблема теперь в следующем: представьте, что вы через месяц открываете этот код и пытаетесь понять, что значит каждое из чисел:

```
<script>  
  console.log(10000 / 0.147);  
  console.log(10000 * 1.01);  
</script>
```

Возможно, вы даже вспомните, но на это уйдёт время.



# ПЕРЕМЕННЫЕ





# Переменные

Чтобы не приходилось вспоминать, в JS есть возможность объявлять переменные. Переменные – это просто удобные имена для хранения и использования информации.

Переменные объявляются с помощью ключевых слов `let` или `const` и выглядит это следующим образом:

```
<script>  
  const input = 10000;  
  const exchangeRate = 0.147;  
  const commission = 0.01;  
  
  const amount = input / exchangeRate;  
  const payment = input * (1 + commission);  
  
  console.log(amount);  
  console.log(payment);  
</script>
```

← привязали к именам значения

теперь можем использовать имя вместо того, чтобы везде писать значение



# Переменные

Давайте разбираться:

1. Ключевое слово `const` говорит, что мы связываем имя со значением всего один раз (это пока будет сложно понять, поэтому пока просто привыкните использовать `const`)
2. Оператор `=` занимается тем, что вычисляет то выражение, которое написано справа, а затем привязывает его к имени, расположенному слева

При этом вместо имён переменных (например, `input`) подставляются те значения, которые к ним "привязаны" (т.е. `10000`).



# Переменные

Давайте посмотрим, как это всё работает в отладчике. Сейчас мы находимся на 3-ей строке (она подсвечена, а значит браузер её ещё не выполнил):



В боковой панели, в разделе **Scope** будут заполняться новые имена по мере того, как вы будете "прошагивать" строки.



# Переменные

Важно: переменные нужны для двух вещей:

1. Давать имена данным (поэтому имена переменных должны быть понятны):
  - имена принято писать на английском, потому что сейчас большинство проектов – международные, и если каждый участник проекта будет писать на своём языке, то они друг друга просто не пойму
  - важно: **summa**, **procent** – это не английский язык!
2. Использовать имена вместо того, чтобы руками менять данные везде: чтобы сосчитать данные не для **10 000**, а для **20 000**, нам достаточно поменять значение только в одном месте



# Имена переменных

Имена переменных принято писать с маленькой буквы, каждое следующее слово начиная с большой: `exchangeRate`.

Обычно в качестве имён используются одно-три слова, не больше.

Общие правила, применимые к именам:

1. Должно быть осмысленным
2. Начинается с буквы (`_`, `$` – не рекомендуется)
3. Содержит буквы, цифры, (`_`, `$` – не рекомендуется)
4. Регистрозависимо (`exchangerate` и `exchangeRate` – это разные имена)

В компаниях, где строго следят за качеством кода, применяют специальные инструменты – линтеры, которые в автоматическом режиме следят за тем, что вы выполняете эти правила (конечно, первое правило они проверить не могут, но в остальном – вполне).



# Переменные

С переменными есть только одна сложность – им трудно придумывать хорошие имена. Но надо стараться, потому что вот такой код никуда не годится\*:

```
const a = 10000;  
const b = 0.147;  
const c = 0.01;  
  
const d = a / b;  
const e = a * (1 + c);  
  
console.log(d);  
console.log(e);
```

Примечание\*: если вы покажете такой код при устройстве на работу, скорее всего, с вами даже не будут разговаривать.



# const vs let

Ключевое слово **const** позволяет всего один раз связать имя и значение (чаще говорят присвоить).

Ключевое слово **let** позволяет "перепривязывать" к имени значения.

В современном мире рекомендуется использовать **const**, а не **let**.

**Q:** почему, ведь **let** позволяет делать больше?

**A:** относитесь к **const** как к одноразовой посуде – её нельзя переиспользовать, если она уже кем-то использована. Это защищает вас от того, чтобы случайно где-то в глубине программы не присвоить имени другого значения.



# Переменные

Хорошие практики:

1. Используйте `const`
2. Не бойтесь создавать переменные (это не дорого)
3. Сначала кладите всё в переменную, а потом уже выводите в консоль:

```
const amount = input / exchangeRate;  
const payment = input * (1 + commission);
```

```
console.log(amount);  
console.log(payment);
```





# ИТОГОВЫЙ КОД

Итак, мы с вами поработали с переменными, оператором присваивания (=) и некоторыми арифметическими операторами. Итоговый код нашей программы для виджета пока выглядит вот так:

```
const input = 10000;  
const exchangeRate = 0.147;  
const commission = 0.01;  
  
const amount = input / exchangeRate;  
const payment = input * (1 + commission);  
  
console.log(amount);  
console.log(payment);
```

Он ещё не имеет интерфейса, но расчёт производится.



# Переменные

Ключевые моменты: переменная – это некоторое имя с привязанным к нему значением. Значение какого типа хранится в переменной определяет то, какие операции с ней (переменной) можно выполнять.

1. Имя
2. Значение
3. Тип\* (определяется значением)

Давайте подробнее поговорим про операторы.



# Операторы



# Операторы

Ключевыми для работы с числами являются арифметические операторы:

1.  $a + b$  – оператор сложения, например,  $10 + 3 = 13$
2.  $a - b$  – оператор вычитания, например,  $10 - 3 = 7$
3.  $a / b$  – оператор деления, например,  $10 / 3 = 3.3333333333333335^*$
4.  $a * b$  – оператор умножения, например,  $10 * 3 = 30$
5.  $a \% b$  – остаток от деления, например,  $10 \% 3 = 1$
6.  $a ** b$  – возведение в степень, например,  $10 ** 3 = 1000$

Примечание\*: в компьютерах "компьютерная" математика, поэтому вещественные числа там всегда неточные.

Важно запомнить: набор операторов фиксирован. Вы не можете добавить свой или "переделать" существующий.



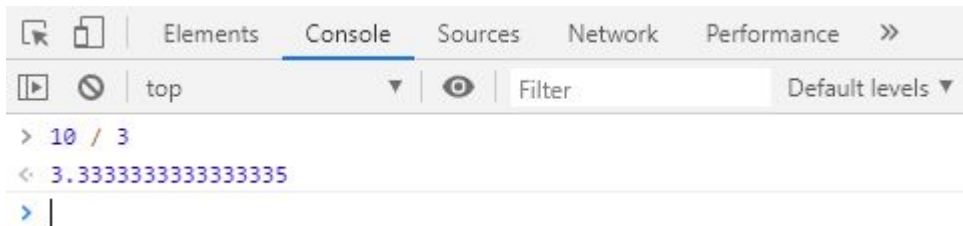
# Операторы

Этих операторов вам будет достаточно, для того, чтобы выполнить ДЗ, остальные мы будем изучать при рассмотрении соответствующих примеров.



# Консоль

Очень удобно небольшие "кусочки" JS проверять в консоли (просто вводите туда выражение и нажимаете на Enter):



# Операторы

Почему мы явно говорили, что с числами можно использовать арифметические операторы? Потому что есть другие типы данных, определяющие свои наборы операторов.



# Типы данных





# Типы данных

Все типы данных делят на две большие категории:

1. Прimitives:

- a. **Boolean** – true/false
- b. **Null** – null (отсутствие значения)
- c. **Undefined** – undefined
- d. **Number** – число
- e. **String** – строка
- f. **Symbol** – символы
- g. **BigInt** - большие целые числа

2. Объекты:

- a. **Object** – объект



# Типы данных

Тип данных определяет то, какие операции можно с этими данными. Остальные типы данных (не числа), мы будем рассматривать следующих лекциях.

Сейчас же давайте посмотрим, что будет, если попробовать в одной операции "смешать" разные типы данных. Посмотрим это на примере самой распространённой связки: числа и строки.



# Строка

Строка – это набор символов. Определяется в виде символов, заключённые в одинарные или двойные кавычки (принято использовать одинарные):

```
const input = '10000';  
const exchangeRate = 0.147;  
const commission = 0.01;  
  
const amount = input / 0.147;  
const payment = input * (1 + 0.01);  
  
console.log (amount);  
console.log(payment);
```

← Теперь это строка, а не число

Но результат в консоли не измениться. Почему? Как можно строку разделить на число?



# Приведение типов

JS всячески старается помочь вам и когда встречает данные разных типов, пытается "привести" их к одному типу. Что значит "привести"? Для всех\* арифметических операторов он пытается из всех типов данных сделать число.

Например, для строки '10000' выполняется преобразование в число 10000, там, где встречается арифметический оператор:

```
const amount = input / 0.147;  
const payment = input * (1 + 0.01);
```

Примечание\*: оператор + со строками превращает всё в строки и склеивает их.



# Приведение типов

Но что если строку нельзя привести к числу, допустим, там написано слово 'много' (мы с вами скоро будем говорить о том, что пользователь может при желании ввести что угодно и вы никак не сможете этому воспрепятствовать):

```
> '10000' / 0.147
< 68027.21088435374
> 'много' / 0.147
< NaN
>
```

Что такое **NaN**? **NaN** – это специальное значение, которое представляет из себя "Не Число" (Not a Number).

В JS достаточно много подобных тонкостей, но на начальном этапе изучения языка лучше запомнить универсальное правило: старайтесь не использовать в одном выражении разные типы данных. Как это сделать, мы с вами посмотрим, когда будем работать с формами.



# Итоги



# Итоги

В этой лекции мы начали знакомиться с JS и даже рассмотрели небольшой практический пример использования. Самое время переходить к JS Level 1, который мы начнём совсем скоро!



# ДОМАШНЕЕ ЗАДАНИЕ





# Орг.моменты

Практикум состоит из 8 обязательных занятий. Начиная с 23 декабря мы выкладываем новые занятия каждый Пн в 10:00 (по Душанбе).

Каждое воскресенье в 23:59 (по Душанбе) дедлайн сдачи домашнего задания.

Если не успеете сдать в срок домашнее задания, тогда этот практикум будет для вас закончен и вы сможете зарегистрироваться на запуск следующего через несколько месяцев.

Все вопросы вы сможете задавать в [Телеграм канале](#).



# Важно

Создайте файл `app.js` (в каталоге `docs`), в котором напишите программу, требуемую в ДЗ.

**Важно:** бот будет искать только `app.js`.



# Важно

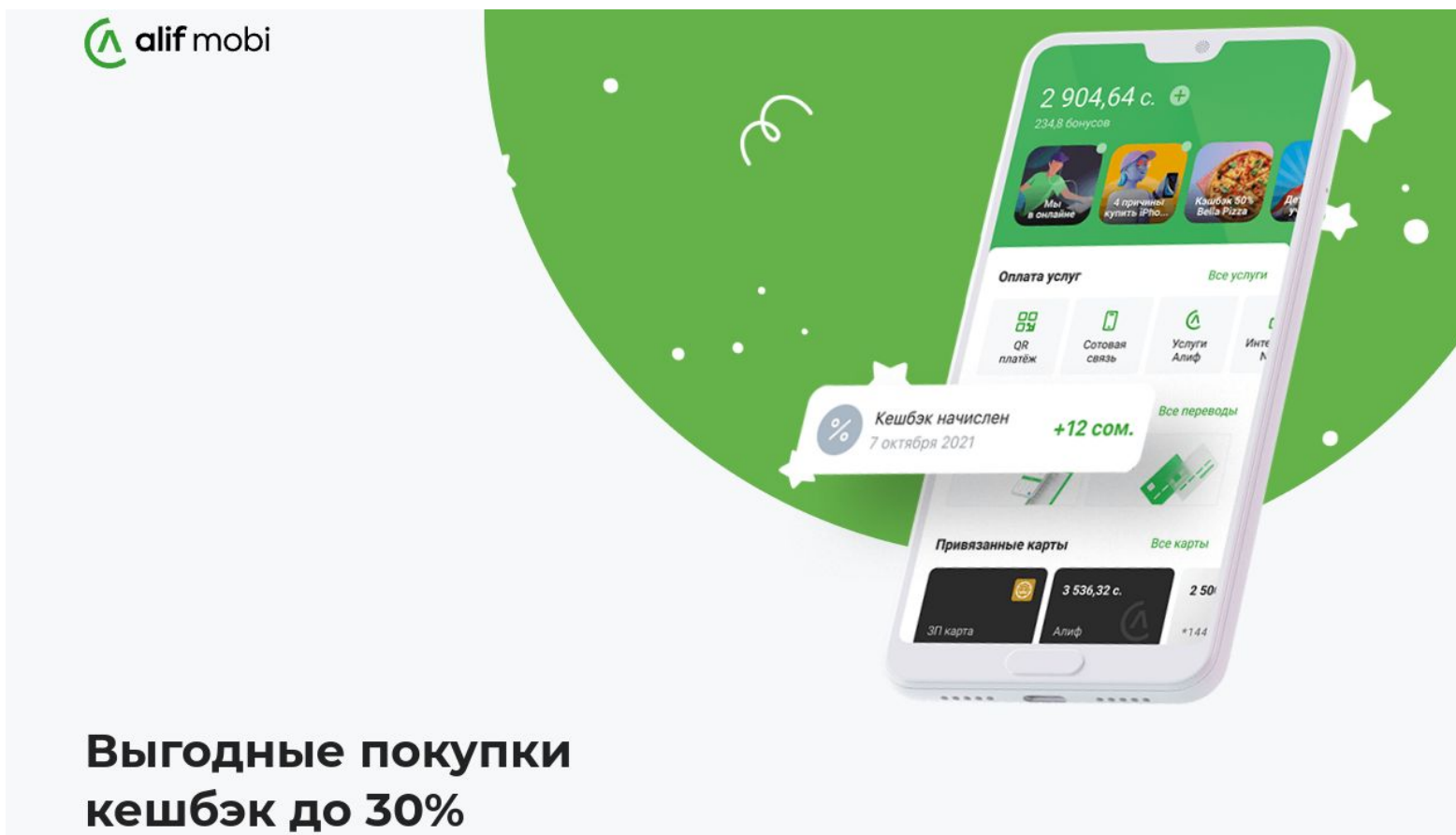
Используйте вот такой документ для подключения [app.js](#):

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Документ</title>
</head>
<body>
  <script src="<u>./app.js</u>"></script>
</body>
</html>
```



# ДЗ: Кэшбэк

Давайте посмотрим на сервис <https://cashback.alif.tj>:



# ДЗ: Кэшбэк

Правила сервиса гарантируют вам с любой покупки получение кэшбэка в зависимости от магазина.

Вам необходимо написать программу, которая исходя из суммы покупки и кэшбэка конкретного магазина высчитывает итоговый кэшбэк. Важно: округлять до сомони не нужно (мы разберём вопросы округления позже).



# ДЗ: Кэшбэк

Чтобы автоматизированная система смогла проверить вашу задачу, необходимо выполнить ряд требований:

1. Переменная для вводимой суммы покупок должна объявляться, как `const input = число;` (например, `const input = 10000`), переменная для размера % кэшбэка магазина как `const storePercent = число;` (например, `const storePercent = 50;` для 50% кэшбэка)
2. Переменная для результатов должна называться `cashback`

```
1  const input = 1000;  
2  const storePercent = 50;  
3  
4  // ваш код  
5  
6  const cashback = ...;  
7  
8  console.log(cashback);
```



# ДЗ: Кэшбэк

**Важно:** бот сначала запустит программу с вашими данными, а затем заменит строку `"const storePercent = 50;"` на `"const storePercent = 30;"`



Спасибо за внимание

**alif skills**

2022г.

